UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/798,936	03/11/2004	Kenneth A. Chupa	AUS920031075US1	6909

32329 7590 06/10/2009

IBM CORPORATION
INTELLECTUAL PROPERTY LAW
11501 BURNET ROAD
AUSTIN, TX 78758

EXAMINER	
LEE, MARINA	

ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
06/10/2009	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/798,936	CHUPA ET AL.	
	Examiner	Art Unit	
	MARINA LEE	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *20 March 2009*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-7 and 21-33* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-7 and 21-33* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Applicant's amendment and response dated March 20, 2009 in responding to the Office Action of December 22. 2008 provided in the rejection of all pending claims 1-7.

Claims 21-33 have been added. Claims 8-20 were previously canceled.

Thus, claims 1-7 and 21-33 are presented for the examination.

2. Applicants' arguments for the claims have been fully considered but they are not persuasive, as will be also addressed under Prior Art's Arguments – rejections section at item (3) below. Thus, the rejection of the claims over prior art in the previous Office Action is maintained with the additional clarification hereon to the Applicants' amended claims, which those claims limitation are either brought from the dependent claims limitation into independent claims or vice versa and accordingly, **THIS ACTION IS MADE FINAL**. See MPEP §706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will

the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Prior Art's Arguments – Rejections

3. Applicants' arguments filed on March 20, 2009 have been fully considered but they are not persuasive.

In response to the rejection to claims 1-7 under 35 U.S.C. § 112, second paragraph, Applicants allege that, "claim 1 states that the interfaces for a deployable software component are compared in the first descriptor file and in the second descriptor file. As discussed in Applicants' Specification, the interfaces for the current Enterprise JavaBeans in the input and output Java Archive (JAR) files are compared. Specification, page 8, lines 24-25. If one or more of these miscompare, then the current Bean is tagged. Specification, page 8, lines 25-26. Hence, different deployable software components are not being compared in different descriptor files. Thus, it would not make sense to add the phrase "in a first descriptor file" or "in a second descriptor file" after the phrase "said first deployable software component" or "said second deployable software component" as requested by the Examiner." – *See Remarks, page 8, last paragraph.*

As of the above assertion, Applicants appear to contend that as of claim 1 recites "the interfaces for a deployable software component are compared in the first descriptor file and in the second descriptor file" to be mean " the interfaces for the current Enterprise JavaBeans in the input and output Java Archive (JAR) files are compared. Specification, page 8, lines 24-25. If one or more of these miscompare, then the current Bean is tagged" with emphasis added. Again, as mentioned in the previous Office action, "if the interfaces for the current

JavaBean" of both "the input JAR files" and "the output JAR files" are being miscompare(not match), the question is "which interface for current JavaBean" as Applicants claims to be "tagged" belong to "the input JAR files" or "the output JAR file".

As in claim 1 merely recites "if the comparing step miscompare for a first deployable software component, tagging said first deployable software component" (with emphasis added), does not disclose, which "base" that "the comparing step" that "the first deployable software component" is going to "tag", when "the interfaces for the deployable software component of a preselected input achieve file and the interface for the deployable software of a preselected output archive file is miscompare (not match)" .

Accordingly, for expedite the process, examiner treats "tagging said first deployable software component" to be "adding components of input file into the output file" during miscomparing (e.g. deployable module of the construct list (input file) was not found in the application deployment descriptor (output file), add (tag) each module in the modules to deploy list of the application descriptor (output file)). Examiner also treats "tagging said second deployable software component" to be "adding components of input file into the output file" during miscomparing (e.g. deployable module of the construct list (input file) was not found in the application deployment descriptor (output file), add (tag) each module in the modules to deploy list of the application descriptor (output file)).

As per claims 1, 21, and 28, Applicants allege that Garms and Spring, taken singly or in combination, do not teach "for each deployable software component in an

preselected input archive file, comparing interfaces for the deployable software component identified in a first descriptor file in said input archive file and a second descriptor file in a preselected output archive file" – See Remarks, page 10, ¶ 1, by meanly insert that "There is no language in the cited passages that teaches comparing interfaces for the deployable software component identified in a first descriptor file in the input archive file and a second descriptor file in a preselected output archive file. Instead, Spring teaches comparing the current data structure to the previous data structure describing the interface for the previous version of the core module. Spring does not teach comparing the interfaces identified in a first descriptor file or a second descriptor file. Neither is there any language in the cited passages that teaches comparing interfaces for the deployable software component identified in a first descriptor file in the input archive file and a second descriptor file in a preselected output archive file for each deployable software component in a preselected input archive file. Garms teaches creating a list of modules to deploy for those deployable modules not found in the deployment descriptors. However, there is no language in either Garms or Springs that teaches performing the comparison step for each deployable software component." – *See Remarks page 11, ¶ 1*, which examiner respectfully disagrees.

Spring teaches "Techniques for maintaining version compatibility between a first module and one or more interacting modules that interact with the first module through an interface. Stored mapping maps instances of a data structure describing the interface at corresponding release times with corresponding version numbers for the first module. A second version number is automatically developed for a second module of the interacting modules based on the mapping. Compatibility is determined based on a

first version number for the first module and the second version number for the second module. These techniques allow the developer of the central module to provide the developers of the interacting modules with a tool to automatically assign version numbers to the interacting module being developed. Automatic assignment of version numbers avoids the tedium and errors associated with manual methods. Furthermore, the automated methods not only ensure compatibility but also discover compatibility with the earliest core module." – *See Spring, at least Abstract, with emphasis added.*

Spring also discloses "In effect, the interface of the network management system comprises two Java interfaces, one Java interface for methods invoked by the core module and implemented in the domain-specific module, and a second Java interface for methods invoked by the domain-specific module and implemented in the core module. As used herein, a software interface includes names and types of either implemented methods or abstract methods or both. When a Java interface is intended it will be explicitly stated as a Java interface." – *See Spring, at least, col. 6: 58-66 with emphasis added.*

Spring further discloses, "In one embodiment, the minor version number is changed automatically. This embodiment uses a current data structure describing the interface for the newest version of the core module and compares the current data structure to the previous data structure describing the interface for the previous version of the core module. If the contents of both data structures are the same, the minor version number is not changed. If the contents of the two data structures differ, the minor version number is incremented by one."— *See at least col. 9: 38-46 with*

emphasis added. Spring furthermore discloses "Data structure 402a describes the interface of the core module at a first release time and version number 403a represents the corresponding version number at that first release time. Data structure 402b describes the interface of the core module at a next release time and version number 403b represents the corresponding version number at that next release time. Ellipses 405 indicate that the mapping 400 may include other data structures and corresponding version numbers for other releases at other times." – *See Spring, at least col. 10: 44-52 with emphasis added.*

As of supra, "a data structure" of Spring describes "the interface" for "the core module (file)" and if "the content" of both data structure (describing the interface) are the same or differ (comparing by using version number). Accordingly, the Applicants' above argument that "there is no teaching of the interface comparing" is not persuasive since "the data structure" of Spring is used to describes/defines "the interface of the core module (component)".

As mentioned in the previous Office action (pages 5-6), It is also to note that Garms does not explicitly disclose comparing interface of each deployable module during the comparing step. However, Spring, in an analogous art, teaches the limitation as see above discussion. Therefore, it would have been obvious to one ordinary skill in the art at the time invention was made to use interface comparing of Spring, in the incremental deployable module of Garms for further optimizing compatibility(e.g., version compatibility) between the construct modules list and the application deployment descriptor as taught in Spring (*e.g., col. 3: 30-44*).

Within claims 1, 21, and 28 limitation, Applicants further contend that " There is no language in the cited passage that teaches tagging a first deployable software component if the comparing step miscompares for the first deployable software component. There is no language in Garms that teaches the comparing step, namely, comparing between the interfaces for the deployable software component in the input archive file and in the output archive file. Instead, Garms teaches determining if deployable modules are identified without corresponding entries in the application's deployment descriptors." – *See Remarks, page 12, ¶ 2 , which examiner respectfully disagrees.*

As mentioned in the above discussion regarding to the claim rejection under 35 U.S.C 112 second paragraph, for expedite the process, examiner treats "tagging said first deployable software component" to be "adding components of input file into the output file" during miscomparing (e.g. deployable module of the construct list (input file) was not found in the application deployment descriptor (output file), add (tag) each module in the modules to deploy list of the application descriptor (output file)) – *See Garms at least col. 5: 1-12 with emphasis added.*

Applicants also allege that "There is no language in the cited passage that teaches tagging a second deployable software component if the comparing step miscompares for the second deployable software component" recites per claims 1, 21, and *28 – See Remarks, page 12, last paragraph, which examiner respectfully disagree.*

As stated above under the claim rejection under 35 U.S.C 112 second paragraph, for expedite the process, examiner treats "tagging said second

deployable software component" to be "adding components of input file into the output file" during miscomparing (e.g. deployable module of the construct list (input file) was not found in the application deployment descriptor (output file), add (tag) each module in the modules to deploy list of the application descriptor (output file)) – See Garms at least col. 5: 1-12 with emphasis added.

As of the forgoing discussion above, Garms in view of Spring does teach the limitation recite per claims 1, 21, and 28.

As per claims 4, 24, and 31, Applicants alleges that "There is no language in the cited passages that teaches introspecting a binary class file for the first deployable software component in the input and output archive files.Neither is there any language in the cited passages that discloses introspecting a binary class file for the first deployable software component in the input and output archive files if the first descriptor file and second descriptor file compare for the first deployable software component. Neither is there any language in the cited passages that discloses tagging the first deployable software component if, in response to the introspection, a signature or return type of an interface of the binary class files miscompare" – *See Remarks, page 15, ¶ 3*, which examiner respectfully disagrees.

Garms teaches if the first descriptor file and second descriptor file compare for the first deployable software component (e.g., each deployable module of the construct module list (input file) was not found in the application deployment descriptor (output file), add (tag) each module in the modules to

deploy list of the application deployment descriptors (output file) – See Garms, at least 5: 1-12 with emphasis added),

As of supra discussion, "a data structure" of Spring describes "the interface" for "the core module (file)" and if "the content" of both data structure (describing the interface) are the same or differ (comparing by using version number). It is further to note that "a data structure describes/defines interface" of Spring in one embodiment is Java such that "The use of Java simplifies the generation of the data structures 430 because the declarations for the fields and routines are stored with the binary class files. This embodiment uses a GNU Bytecode Java package to access the declarations in the binary class files." – See Spring, at least col. 13: 13-20 with emphasis added. Spring also discloses "another embodiment 470 of a data structure describing the interface of a core software module. In this embodiment, the unique signatures of the routines in the interface are not used directly in substructures 450 but are mapped through a hash function to unique codes stored in integers 460. The hash function provides a unique integer for each unique function signature." – See Spring, at least col. 13: 25-40 with emphasis added. Spring further discloses "According to one embodiment for software modules, a release of a core module is compatible with an interacting module if all the fields in substructures 435 and routines in substructures 450 referenced by the interacting module appear in the data structure 402 representing the core module interface. For example, interacting module 112a is compatible with both core module versions "2" and "32" because all three substructures in Table 2 appear in the data structures corresponding to both versions of the core module in Table 1. Also, abstract routines in the core module are complimentary implemented routines in the

interacting module 112a, while implemented routines in the core module are complementary abstract routines in the interacting module 112a." – See Spring, at least col. 15: 19-33 with emphasis added. Within the quotation, Spring in combined with Garms does teach the claims 4, 24, and 31 limitation.

As per claims 3, 23 and 30, applicants contents that Garms, Spring, and Cohen taken alone or in combination does not teach or suggest "if the first descriptor file and a second descriptor file compare for the first deployable software component, comparing a size of a binary class file for the first deployable software component n the input and if the size said binary class files miscompare, tagging the fist deployable software component"— See Remarks, page 21, ¶5, by mainly state that "There is no language in the cited passages that teaches comparing a size of a binary class file for the first deployable software component in the input and output archive files. Neither is there any language in the cited passages that teaches comparing a size of a binary class file for the first deployable software component in the input and output archive files if the first descriptor file and second descriptor file compare for the first deployable software component. Neither is there any language in the cited passages that teaches tagging the first deployable software component. Neither is there any language in the cited passages that teaches tagging the first deployable software component if the size of the binary class files miscompare." and "What is the rational connection between comparing a size of a binary class file for the first deployable software component (missing claim limitation) and identifying differences of the deployable module between the construct modules list and theapplication deployment descriptor? Hence, the Examiner's rationale does not provide

reasons that the skilled artisan,

confronted with the same problems as the inventor and with no knowledge of the claimed invention, would modify Garms to include the above-cited missing claim limitation of claims 3, 23 and 30" – *See Remarks, page 22, ¶ 3, and page 25, ¶ 2-3* which examiner respectfully disagrees.

As mention in the previous Office Action (pages 7 & 8) , modified Garms and Spring discloses tagging the first deployable software component if the first descriptor file and second descriptor file compares for the first deployable software component, (*e.g., each deployable module of the construct module list (input file) was not found in the application deployment descriptor (output file), add (tag) each module in the modules to deploy list to the application deployment descriptors – See Garms, at least 5: 1-12 with emphasis added*).

It is noted that, however, modified Garms with Spring does not explicitly disclose comparing a size of a binary class file for the first deployable software component. But, Cohen, in an analogous art, teaches "The delta file identifies the differences between binary file data A and binary file data B. In other words, applying the delta file to file data A yields file data B (or visa versa). At 204, the delta binary file is compressed. At 206, the size of the compressed delta binary file is compared to compressed binary file data B. Based on this comparison, a determination is made at 208 to determine whether the delta binary file is acceptable. This determination may simply include a comparison of the size of the compressed delta binary file as compared to the size of binary file data B which the delta binary file is intended to replace or it may include other

comparisons such as restoration time. If the size of the delta binary file is smaller (e.g., at least 25% smaller), this means that the combination of file data A and the delta binary file will be smaller than the combination of file data A and file data B. Thus, the delta binary file is acceptable and at 210 file data A and the delta file are stored as part of the volume image because they would be smaller that file data A plus file data B. If the size of the delta binary file is near the size of or larger that file data B, this means that the combination of file data A and the delta binary file will be larger than the combination of file data A and file data B. Thus, the delta binary file is unacceptable and at 212 file data A and file data B are stored as part of the volume image because they would be smaller that file data A plus the delta binary file. – *See Cohen at least [0091], Fig. 2, and associated text, with emphasis added.*

It would have been obvious to one ordinary skill in the art at the time invention was made to use binary size comparing in binary file of Cohen, in the incremental deployable module of the modified Garms with Spring for efficiency deployment in the deployable module between the construct modules list and the application deployment descriptor due to its smaller size format that help in reducing the chances of wasting digital storage media as taught in Cohen (*e.g., [0091] & [0004]*).

As to the rest of claims which are depend upon the independent claims 1, 21, and 28 respectively are also fail to be found persuasive for at least the above stated reason.

Claim Rejections - 35 USC § 112

4. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

5. Claims 1-7 and 21-33 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

As to claim 1,

Lines (6-7), recite the limitation "tagging said first deployable software component". There is insufficient antecedent basis for this limitation in the claim since it is unclear of what (e.g. *deployable software component identified in a first descriptor file or in a second descriptor file)* being tagged for "said *first* deployable software component";

Lines (8-9), recite the limitation "tagging said second deployable software component". There is insufficient antecedent basis for this limitation in the claim since it is unclear of what (e.g. *deployable software component identified in a first descriptor file or in a second descriptor file)* being tagged for "said *second* deployable software component".

As of the forging discursion above; therefore, appropriate corrections are required.

Accordingly, for expedite the process, examiner treats "tagging said first deployable software component" to be "adding components of input file into the output file" during miscomparing (e.g. deployable module of the construct list

(input file) was not found in the application deployment descriptor (output file), add (tag) each module in the modules to deploy list of the application descriptor (output file)). Examiner also treats "tagging said second deployable software component" to be "adding components of input file into the output file" during miscomparing (e.g. deployable module of the construct list (input file) was not found in the application deployment descriptor (output file), add (tag) each module in the modules to deploy list of the application descriptor (output file)).

Claims 21 and 28 are also rejected under similar reasons as of claim 1 above.

Claims 2-7, 22-27, and 29-33 are also rejected to for not meeting the rejection of the base claims 1, 21, and 28 respectively. Therefore, they are also rejected for the same reason.

Claim Rejections - 35 USC § 103

6. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

7. Claims 1, 2, 4, 7, 21, 22, 24, 2-29, and 31 are rejected under 35 U.S.C. 103(a) as being unpatentable over Garms et al. (US 7,296,255 B2 of record – hereinafter Grams) in view of Spring et al. (US 6,971,093 B1 of record – hereinafter Spring).

As per claims 1, 21, and 28, Garms discloses a method for selectively deploying enterprises software comprising:

for each deployable software component in a preselected input file (*e.g., a deployable module of a construct modules list (preselected input file) – See at least col. 4: 63-67 with emphasis added)*, comparing the deployable software component identified in a first descriptor file in said input archive file and a second descriptor in a preselected output file (e.g., a deployable module of application's deployment descriptors (output file) – see at least col. 5: 1-4 with emphasis added) – (*e.g., each deployable module of the construct module list (input file) is being compared with each deployable module of the application deployment descriptor (output file) – See at least 5: 1-12 with emphasis added*); if the comparing step miscompares for a first deployable software component, tagging said first deployable software component;

if the comparing step miscompares for a first deployable software component, tagging said first deployable software component (*e.g., each deployable module of the construct module list (input file) was not found in the application deployment descriptor (output file), add (tag) each module in the modules to deploy list to the application deployment descriptors – See at least 5: 1-12 with emphasis added*); and

if the comparing step miscompares for a second deployable software component, tagging said second deployable software component (*e.g., each deployable module of the construct module list (input file) was not found in the application deployment descriptor (output file), add (tag) each module in the*

modules to deploy list to the application deployment descriptors – See at least 5: 1-12 with emphasis added); and

deploying each tagged deployable software component -- *se at least col. 6: 8-9 and step 232 (fig. 2c)*.

It is noted that Garms does not explicitly discloses that (e.g., the construct modules list) and (the application deployment descriptor) are store as a file. However, it would have been implied that (the construct list) and (the application deployment descriptor) must be stored (e.g., as a file) somehow for comparing step to happen. It is further to not that Garms does not explicitly discloses that the file of (*e.g., construct list and the application deployment descriptor*) are archive. However, it is well known that archive files are used to collect multiple data files together into a single file for easier portability and storage. Thus, it would have been obvious to one ordinary skill in the art at the time invention was made to use the archive file for storing (*e.g. the construct modules list and the application deployment descriptor*) of Garms due to the using archive file.

It is also to note that Garms does not explicitly disclose comparing interface of each deployable module during the comparing step. However, Spring, in an analogous art, teaches maintaining compatibility of a software core module and an interacting module such that "*in one embodiment, the minor version number is changed automatically. This embodiment uses a current data structure describing the interface for the newest version of the core module and compares the current data structure to the previous data structure describing the interface for the previous version of the core module. If the contents of both data*

structures are the same, the minor version number is not changed. If the contents of the two data structures differ, the minor version number is incremented by one." – See Spring, at least col.9: 38-46, col. 10: 19-26, and col. 13: 49-63. col. 6: 58-66, col. 9: 38-46, abstract and col. 10: 44-52 t with emphasis added.

It would have been obvious to one ordinary skill in the art at the time invention was made to use interface comparing of Spring, in the incremental deployable module of Garms for further optimizing compatibility(e.g., version compatibility) between the construct modules list and the application deployment descriptor as taught in Spring (*e.g., col. 3: 30-44*).

Further regarding to claim 21, Spring discloses a computer product embodied in a computer-readable medium (*e.g. storage device 610 – see Spring at least col. 17: 50*) having computer program executable code recorded thereon for implementing method as of claim 1 above.

Further regarding to claim 28, Spring discloses a data processing system *(e.g., computer system 600 – see Spring, at least col. 17: 28)* for implement method as of claim 1 above**.**

As per claims 2, 22, and 29, Garms further discloses wherein tagging a deployable software component comprises storing a name of the displayable software component in a file (*e.g., module tag name of Deployment Descriptor such as application.xml and web-logic-application.xm – See at least col. col. 6: 19-65*).

As per claims 4, 24, and 31, modified Garms with Spring discloses further comprising:

if the first descriptor file and second descriptor file compare for the first deployable software component (*e.g., each deployable module of the construct module list (input file) was not found in the application deployment descriptor (output file), add (tag) each module in the modules to deploy list to the application deployment descriptors – See Garms, at least 5: 1-12 with emphasis added*), introspecting a binary class file (e.g. The use of Java simplifies the generation of the data structures 430 because the declarations for the fields and routines are stored with the binary class files. This embodiment uses a GNU Bytecode Java package to access the declarations in the binary class files." – See Spring, at least col. 13: 13-20 with emphasis added) for the first deployable software component in the input and output archive files; and

if, in response to the introspection, a signature or return type of an interface -- (*see Spring, at least col. 13: 1*-40 and col. 15: 19-33 with emphasis added) .of said binary class files miscompare, tagging the first deployable software component

As per claims 7 and 27, modified Garms with Spring does not explicitly disclose wherein the comparing, tagging and deploying steps are performed in response to an execution of a build script invoking a selective deployer utility. However, Garms deployment descriptor to overcome manually work (*see at Garms, at least col. 3: 36-59 with emphasis added*). Thus, it would have been obvious to one ordinary skill in the art at the time invention was made to realize

that scripting must have been invoked for incremental deployment all files under development of Garms to not manual (automatically implemented) implement.

8. Claims 3, 23, and 30 are rejected under 35 U.S.C. 103(a) as being unpatentable over Garms et al. (US 7,296,255 B2) in view of Spring et al. (US 6,971,093 B1), and in further view of Cohen et al (US 2004/0034849 A1 of record – hereinafter Cohen).

As per claims 3, 23, 30, modified Garms and Spring discloses tagging the first deployable software component if the first descriptor file and second descriptor file compares for the first deployable software component, (*e.g., each deployable module of the construct module list (input file) was not found in the application deployment descriptor (output file), add (tag) each module in the modules to deploy list to the application deployment descriptors – See Garms, at least 5: 1-12 with emphasis added*).

It is noted that, however, modified Garms with Spring does not explicitly disclose comparing a size of a binary class file for the first deployable software component. But, Cohen, in an analogous art, teaches "The delta file identifies the differences between binary file data A and binary file data B. In other words, applying the delta file to file data A yields file data B (or visa versa). At 204, the delta binary file is compressed. At 206, the size of the compressed delta binary file is compared to compressed binary file data B. Based on this comparison, a determination is made at 208 to determine whether the delta binary file is acceptable. This determination may simply include a comparison of the size of the compressed delta binary file as compared to the size of binary file data B

which the delta binary file is intended to replace or it may include other comparisons such as restoration time. If the size of the delta binary file is smaller (e.g., at least 25% smaller), this means that the combination of file data A and the delta binary file will be smaller than the combination of file data A and file data B. Thus, the delta binary file is acceptable and at 210 file data A and the delta file are stored as part of the volume image because they would be smaller that file data A plus file data B. If the size of the delta binary file is near the size of or larger that file data B, this means that the combination of file data A and the delta binary file will be larger than the combination of file data A and file data B. Thus, the delta binary file is unacceptable and at 212 file data A and file data B are stored as part of the volume image because they would be smaller that file data A plus the delta binary file. – *See Cohen at least [0091], Fig. 2, and associated text, with emphasis added.*

It would have been obvious to one ordinary skill in the art at the time invention was made to use binary size comparing in binary file of Cohen, in the incremental deployable module of the modified Garms with Spring for efficiency deployment in the deployable module between the construct modules list and the application deployment descriptor due to its smaller size format that help in reducing the chances of wasting digital storage media as taught in Cohen (*e.g., [0091] & [0004]*).

9. Claims 5 , 6 , 25, 26, 32, and 33are rejected under 35 U.S.C. 103(a) as being unpatentable over Garms et al. (US 7,296,255 B2) in view of Spring et al.

(US 6,971,093 B1), and in further view of Kovacs et al., (US 2004/0158571 A1 of record, hereinafter – Kovacs).

As to claims 5, 25, 32, it is noted that modified Garms and Spring does not expressively disclose further comprising: opening said preselected output archive file; and if the step of opening the preselected output archive fails, tagging each deployable software component in the input archive file. However, Kovacs, in an analogous art, teaches validate deployment descriptor information (i.e. validator 302) to locate errors within deployment descriptor files (e.g., incorrect CMP field name, etc.) by displaying or highlighting the error message to the user. –See (Kovacs, 302, Fig. 3, and page 2, [0020] & [0021]).

It would have been obvious to one of ordinary skill in the art at the time invention was made to have been motivated to apply error validator 302 of Kovacs in deployment descriptor of Garms for assisting developers in user friendly interface (e.g., pop-up window) to identify the input errors and offer the suggesting solution to those errors as taught in Kovacs (e.g., page 2, [0021]).

As per claims 6, 26, and 33, Kovacs further discloses wherein the sep of tagging each deployable software component is performed in response to the stop of opening the preselected output archive throwing an exception (see Kovacs, page 2, [0020]&[0021]).

Conclusion

10. The prior art made of record and not relied upon is considered pertinent to application disclosure.

11. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Marina Lee whose telephone number is (571) 270-1648. The examiner can normally be reached on M-F (11:00 am to 7: 30 pm) EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on (571) 272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/M. L./
Examiner, Art Unit 2192

/Tuan Q. Dam/
Supervisory Patent Examiner, Art Unit 2192